



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Introducción a la Programación IIC1103

Prof. Ignacio Casas
icasas@ing.puc.cl

Tema 06 – Programación Orientada a Objetos
Parte 1.1

Colaboración de Mauricio Arriagada

Agenda

1. Modelación (Programación) con Objetos
2. Tipos de datos y Clases
3. Python: Creando Clases
4. Python: Creando Objetos de una Clase
5. Ejercicios

Introducción:

Objetos: una representación del mundo real

Con los programas computacionales **representamos** (**modelamos**) situaciones de la vida real.

Podemos **modelar** (**representar**) diversas situaciones de: matemática, física, química, biología, ingeniería, finanzas, juegos, transporte, transacciones bancarias, clientes de una empresa, alumnos de una universidad, propiedades de una inmobiliaria, procesos deportivos, etc.

Programación funcional: lo que hemos hecho hasta ahora; programas (modelos) con estructura basada en funciones.

Una forma más eficiente (y de estado del arte) para hacer esto es con **objetos**

Programación orientada a objetos: Cuando representamos (modelamos) una situación con **objetos**.

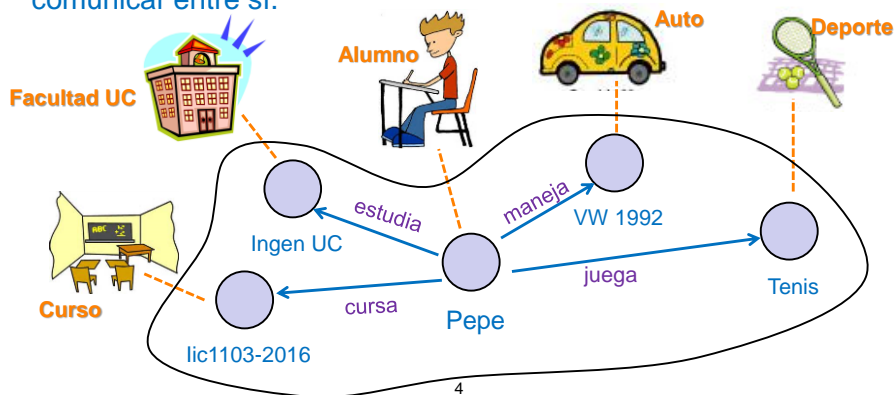
Python facilita la definición y manipulación de **objetos**.

3

Introducción:

Objetos: una representación del mundo real

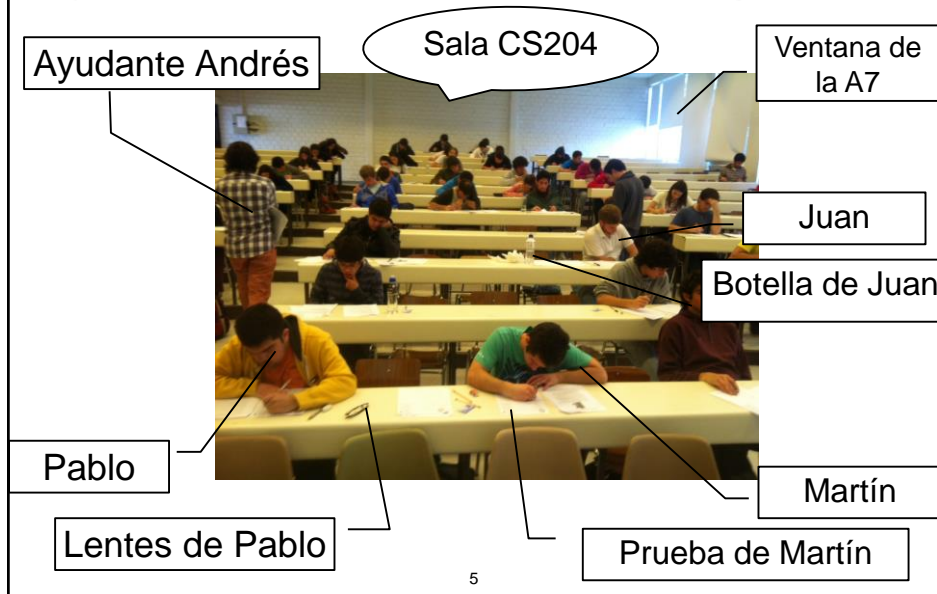
Un modelo orientado a **objetos** representa una cierta situación (estática o dinámica) como un **conjunto de objetos relacionados-asociados** entre sí. Los objetos tienen **atributos** y **comportamientos**. Se agrupan en **clases** y se pueden comunicar entre sí.



4

Introducción

Objetos: podemos representar el mundo **tangible**



5

Introducción:

Objetos: una representación del mundo real

También podemos usar objetos para representar conceptos

intangibles, tales como:

tiempo, sincronización, números, notas, procesos, transacciones, eventos, demandas, viajes, etc.

Definimos un **objeto** en base a sus **atributos** (características) y su **comportamiento** (funciones-métodos).

Por ejemplo, el auto de Pepe tiene **atributos** tales como:

marca, modelo, año de fabricación, color, número de motor, etc.

Y **comportamientos** tales como:

enciende, acelera, viaja, para, choca, etc.

Para gatillar los comportamientos de un objeto, se le deben enviar **mensajes** o **invocaciones** (como a una función).

6

Propiedades de los objetos en Python

Valores, Datos

definen lo que es

Un **objeto** en Python posee **atributos** y **comportamientos**

lo que puede hacer

Métodos-
Funciones

Objetos que comparten similares atributos y comportamientos, pertenecen a una misma **clase**.
Ejemplo: Pepe es de clase Alumno.
VW 1992 es de clase Auto.

7

Clases: Todo objeto es de un cierto tipo o clase

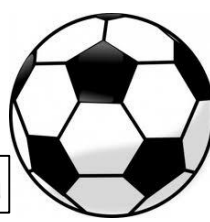
- Cada objeto tendrá los atributos y comportamientos (acciones que podrá realizar) dependiendo de la clase a la que pertenezca.

Balón Cafusa



Objeto

Balón de Fútbol



Clase

Tipo

Instancia

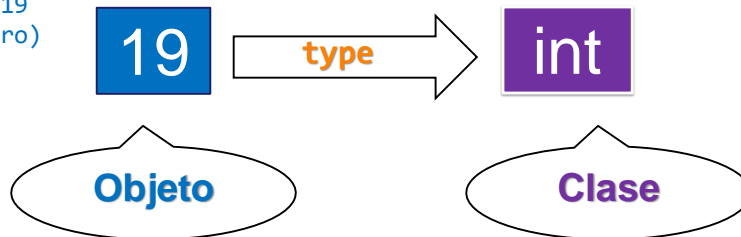
8

Clases en Python: Función type

- La función `type` retorna la clase/tipo del objeto que le entregamos como parámetro.

Ejemplo:

```
numero = 19
type(numero)
```



19 es un objeto de tipo o clase **int**
19 es una instancia de la clase **int**

La variable **numero**:
 almacena el objeto **19**
 (es un "alias" de, apunta a)

9

Clases (tipos de datos) pre-definidas en Python: Básicas y compuestas

- En Python existen tipos de datos (clases) pre-definidos básicos y compuestos:

- Básicos: **enteros, reales, booleanos**

```
mi_numero = 19
print(type(mi_numero))
<class 'int'>
```

- Compuestos:
strings, listas, tuplas, diccionarios

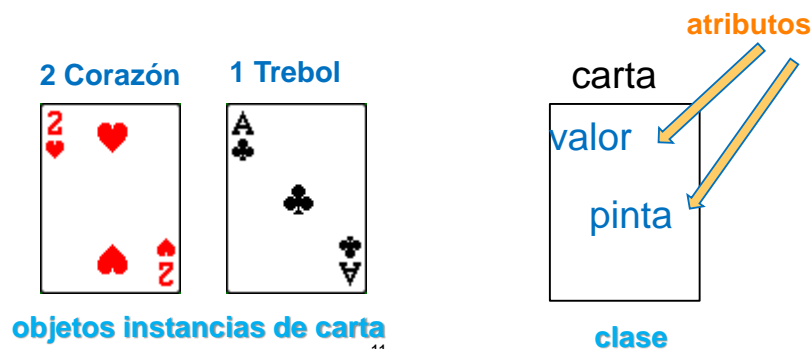
```
mi_lista = [1,2,3,4]
print(type(mi_lista))
<class 'list'>
```

10

Clases:

Tipos de datos (clases) definidos por el usuario

- Además de los **tipos de datos (clases) pre-definidos** por Python, podemos crear nuestras propias **clases**.
- Por ejemplo: Una clase (tipo de dato) que representa las cartas de una baraja inglesa.



11

Clases: Creando una clase

- Python utiliza la palabra **class** para crear objetos.
- Se define en forma similar a una función:
 - Encabezado **class nombre_clase:**

cuerpo o bloque de sentencias que puede contener variables (atributos) y métodos (comportamientos).

```
class Carta:
    # aquí ponemos los atributos (variables)
    # y comportamientos (funciones/métodos)
    # que definen a la clase "Carta"
```

12

Clases: Atributos

- Los atributos de una clase definen las características de cada objeto (instancia) de esa clase. En Python se definen con **variables**, por ej, en la clase **Carta**: **pinta** y **valor**

```
class Carta:
    pinta = "blanco"
    valor = 0
# comienza el programa principal
b = Carta()
print("tipo-clase de b: ", type(b))
print("pinta = ", b.pinta, " valor =", b.valor)
print("Si imprimo b: ", b)
print("Si imprimo Carta: ", Carta)
print("Es b instancia de Carta? ", isinstance(b, Carta))
print("Es 21 instancia de int? ", isinstance(21, int))
```

La creación del objeto (que se guarda) en **b** se llama **instanciación**. Dicho **objeto** es una **instancia** de **Carta**.

Clases: Atributos

Resultado del programa anterior:

```
>>>
tipo-clase de b: <class '__main__.Carta'>
pinta = blanco valor = 0
Si imprimo b: <__main__.Carta object at 0x02EA83F0>
Si imprimo Carta: <class '__main__.Carta'>
Es b instancia de Carta? True
Es 21 instancia de int? True
>>>
```

Si queremos crear nuevos objetos de la misma clase pero con otros atributos, podríamos hacer:

```
b = Carta()
print("pinta b = ", b.pinta, " valor b =", b.valor)
c1 = Carta()
c1.pinta = "♥"
c1.valor = 7
print("pinta c1 = ", c1.pinta, " valor c1 =", c1.valor)
c2 = Carta()
c2.pinta = "♣"
c2.valor = "K"
print("pinta c2 = ", c2.pinta, " valor c2 =", c2.valor)
```

Clases: Atributos

Resultado del programa anterior:

```
>>>
pinta b = blanco valor b = 0
pinta c1 = ♥ valor c1 = 7
pinta c2 = ♣ valor c2 = K
>>> |
```

Nota: si creamos y guardamos en **c3** un nuevo objeto con los mismos atributos del objeto en **c1** (pinta = "♥", valor = 7), estos serán objetos distintos.

c1 != c3

c1.pinta == c3.pinta

c1.valor == c3.valor

Pero si hacemos: **c4 = c1**

Las variables **c4 y c1** se referirán ("alias", apuntan) al mismo y único objeto creado inicialmente en **c1**.

15

Clases: Atributos y Métodos

Claro que así es bastante "fome" hacer la creación (instanciación) de objetos de clase "Carta".

Existe en Python una forma más elegante y eficiente para hacerlo:

**agregando en la definición de la clase
un método (función) especial: `__init__()`**

también conocido como "constructor"

OJO: son dos "`__`" (underscore) antes y después de **init**

16

Clases: Método “constructor” `__init__`

`__init__()` es un método especial que permite asignar valores a los atributos de un nuevo objeto dentro de la clase.

```
class Carta:
    pinta = "blanco"
    valor = 0
    #
    def __init__(self, mi_pinta, mi_valor):
        self.pinta = mi_pinta
        self.valor = mi_valor
```



Y podemos
crear objetos así:

```
# programa principal
c1 = Carta ( "♥", 7)
c2 = Carta ( "♣", "K")
c3 = Carta ("blanco", 0)
```

OJO:

```
c4 = Carta () # ahora es inválido; genera un error porque __init__
               # necesita valores para los parámetros.
```

17

Clases: Atributos y método “constructor” `__init__`

OJO: En el método `__init__()` estamos definiendo los atributos que tendrán los objetos de esta clase. Ya no es necesario (ni tiene sentido) ponerlos como variables antes del método. Es bueno dejarlos como comentarios con `#` o con `"""`.

```
class Carta:
    """ atributos de esta clase:
        pinta, valor """
    def __init__(self, mi_pinta, mi_valor):
        self.pinta = mi_pinta
        self.valor = mi_valor
```



También podríamos nombrar los parámetros del método
`__init__` igual que los nombres de los atributos:

```
def __init__(self, pinta, valor):
    self.pinta = pinta
    self.valor = valor
```

18

Clases: Creando objetos / instancias


Si queremos crear un objeto en blanco, ya no podemos invocar la clase **Carta** sin argumentos, porque agregamos el método `__init__`. Pero podemos usar parámetros "por defecto":

La triple `"""` es equivalente al `#` pero hay que terminar también con triple `"""`

```
class Carta:
    """ atributos: pinta, valor """

    def __init__(self, pinta = "blanco", valor = 0):
        self.pinta = pinta
        self.valor = valor

    """ programa principal
        instanciando (creando) 3 objetos de la clase carta """
    mi_carta_1 = Carta("Trebol", 1)
    mi_carta_2 = Carta("Corazon", 2)
    Z = Carta()
```



19

Clases: Creando objetos / instancias

Dado que los atributos se definen normalmente con el método `__init__` solo tendrá sentido definir atributos fuera del método, cuando ellos se refieran a características que poseerán todos los objetos de la clase. Por ejemplo:

```
class Carta:
    # atributos: pinta, valor, marca
    marca = "Condorito"
    def __init__(self, pinta="blanco", valor=0):
        self.pinta = pinta
        self.valor = valor
    # comienza el programa principal
    # instanciando (creando) tres objetos de la clase Carta
    a = Carta("♥", 7)
    print("pinta a =", a.pinta, " valor a =", a.valor, " marca a:", a.marca)
    b = Carta("♣", "K")
    print("pinta b =", b.pinta, " valor b =", b.valor, " marca b:", b.marca)
    z = Carta()
    print("pinta z =", z.pinta, " valor z =", z.valor, " marca z:", z.marca)
```

Al definir la marca fuera del `__init__` no necesitamos incluirla cuando creamos (instanciamos) los objetos en las variables a, b, z.

Resumiendo: para crear objetos / instancias

- Para crear (instanciar) un objeto de una clase:

variable = Nombre_Clase (param1, param2, ...)

- Por ejemplo:

```
class Carta:
    """ atributos: pinta, valor
    """
    def __init__(self, pinta, valor):
        self.pinta = pinta
        self.valor = valor

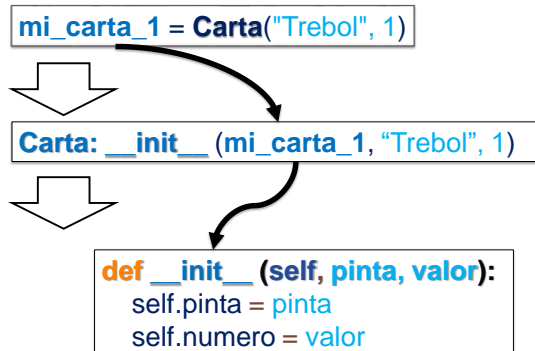
""" instanciando (creando) 2 objetos de la clase carta """
mi_carta_1 = Carta("Trebol", 1)
mi_carta_2 = Carta("Corazon", 2)
```

Los objetos en
mi_carta_1
mi_carta_2
son instancias
de la clase **Carta**.

21

Clases:

¿Internamente cómo se crean los objetos?



22

Clases: Agregando Comportamientos (Métodos)

Para agregar un comportamiento, creamos un método (función) dentro de la clase, asegurando que tenga como primer parámetro **self**

```
class Carta:
    """ atributos: pinta, valor """
    def __init__(self, pinta, valor):
        self.pinta = pinta
        self.valor = valor

    def cambiar(self, pinta, valor):
        self.pinta = pinta
        self.valor = valor
```

Los métodos se definen como funciones dentro de una clase.

23

Clases ¿Cómo se invocan los métodos de una clase?

- Para invocar a un método de la clase, primero tenemos que crear un objeto y luego llamamos al método.

```
class Carta:
    # atributos: pinta, valor
    #
    def __init__(self, pinta, valor):
        self.pinta = mi_pinta
        self.valor = valor
    def cambiar(self, nueva_pinta, nuevo_valor):
        self.pinta = nueva_pinta
        self.valor = nuevo_valor

#programa principal
#instancia de un objeto de la clase carta que se llama mi_carta_1t
```



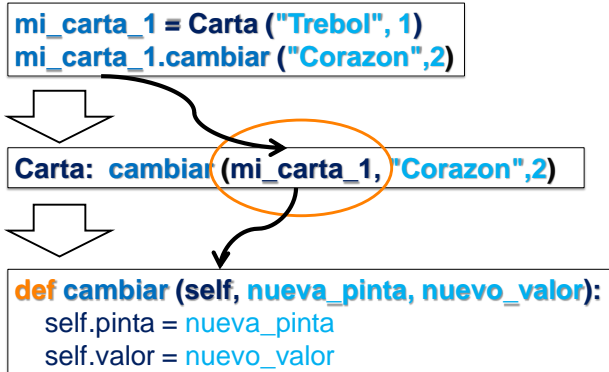
```
mi_carta_1 = Carta ("Trebol", 1)
mi_carta_1.cambiar ("Corazon", 2)
```

Igual que los
métodos
de strings,
listas y tuplas.

24

Clases

¿Cómo funciona la invocación del método?



25

Clases: Entendiendo cómo se instancian los objetos

- Cuando se invoca un método de un objeto, Python se encarga de que el **primer argumento** sea la instancia del objeto que invoca.
- Esto explica que **self** siempre debe ser el primer parámetro del método que se define en una clase.



26

Objetos: ¿Qué es un objeto?

Un objeto es una instancia de una clase, y se “almacena” en la memoria del computador.

```
mi_carta_1 = Carta("Trebol", 1)
print (mi_carta_1t)

<__main__.Carta object at 0x100691050>
```

0x100691048	h
0x100691049	3,1416
0x100691050	
	"Trebol"
0x100691060	1
0x100691065	...
0x100691070	12

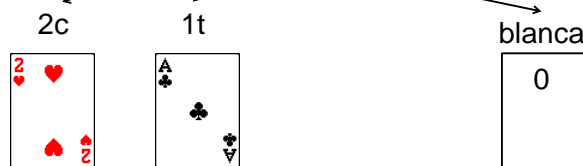
La variable **mi_carta_1** es el “alias” de la dirección del objeto en la memoria del computador.

27

Objetos: ¿Qué es un objeto?

Un objeto es una instancia de una clase, y “existe” en la memoria del computador.

```
class Carta:
    # atributos: pinta, valor
    #
    #
    def __init__(self, pinta, valor):
        self.pinta = pinta
        self.valor = valor
```



28

Objetos: ¿Cómo se accede a los atributos de un objeto?

A través del nombre del objeto seguido de “.” y el nombre del atributo

```
class Carta:
    # atributos: pinta, valor
    #
    def __init__(self, pinta, valor):
        self.pinta = pinta
        self.valor = valor
#
mi_carta_3 = Carta("Trebol", 8)
mi_carta_5 = Carta("Diamantes", 7)
mi_carta_1 = Carta("Corazones", 4)
#
print(mi_carta_5.pinta) → Trebol
print(mi_carta_5.valor) → 7
#
print(mi_carta_1.pinta) → Corazones
print(mi_carta_1.valor) → 4
```

OJO:

Los objetos son **mutables**: podemos cambiar los valores de sus atributos.

Hemos visto que las **listas** también son mutables. Los **strings** no lo son.

Objetos: Copia (replicación) de objetos (incorrecta)

En el siguiente fragmento de código, ¿Por qué cambia el valor de mi_carta_1.numero ?

```
mi_carta_1 = Carta( "Trebol" , 1)
mi_carta_2 = mi_carta_1
mi_carta_2.cambiar( "Trebol" ,7)
print(mi_carta_1, mi_carta_1.numero)
print(mi_carta_2, mi_carta_2.numero)
```

```
<__main__.Carta object at 0x100691090> 7
<__main__.Carta object at 0x100691090> 7
```

Ambas variables apuntan al mismo objeto en la posición 0x100691090. Recuerda que se puede considerar una variable como un “alias” de un objeto.

Objetos: Copia (replicación) de objetos (correcta)

En el siguiente fragmento de código, ¿Por qué NO cambia el valor de `mi_carta_1.numero` ?

```
mi_carta_1 = Carta ("Trebol", 1)
mi_carta_3 = Carta ("Trebol", 1)
mi_carta_3.cambiar ("Corazon",3)
print(mi_carta_1, mi_carta_1.numero)
print(mi_carta_3, mi_carta_3.numero)

<__main__.Carta object at 0x100691090> 1
<__main__.Carta object at 0x1006917d0> 3
```

Ambas variables apuntan a distintos objetos con posiciones de memoria distintas.

La librería **copy** tiene una función **copy** que permite copiar cualquier objeto: `copy.copy(mi_carta_1)`

31



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Introducción a la Programación IIC1103

Prof. Ignacio Casas
`icasas@ing.puc.cl`

Tema 09 – Programación Orientada a Objetos
Parte 1

Colaboración de Mauricio Arriagada